CprE 450/550X
Distributed Systems and Middleware

Inter-process Communication

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

Jan. 28, 2003

Readings for Today's Lecture

- References
  - **Chapter 2 of "Distributed Systems: Principles and Paradigms"**
  - **Chapter 4 of "*Distributed Systems: Concepts and Design*"**
  - **Chapter 14 & Chapter 15 of "Advanced Programming in the UNIX Environment"**

3

Interprocess Communication

- Primitives
- Message Passing: issues
- Communication Schemes

4

Interprocess Communication (IPC)

lack of shared memory → communicate by sending messages

Primitives for interprocess communication

- message passing
  - **the RISC among the IPC primitives**
- remote procedure call (RPC)
  - **process interaction at language level**
  - **type checking**
- transactions
  - **support for operations and their synchronization on shared objects**

5

Message Passing

- The primitives:

```
send expression_list to destination_identifier;

receive variable_list from source_identifier;
```

- Variations:

guarded receive:

```
receive variable_list from source_id when B;
```

selective receive:

```
select
  receive var_list from source_id1;
 |receive var_list from source_id2;
 |receive var_list from source_id3;
end
```

6

Semantics of Message-Passing Primitives

- blocking *vs.* non-blocking
- buffered *vs.* unbuffered
- reliable *vs.* unreliable
- fixed-size *vs.* variable-size messages
- direct *vs.* indirect communication

7

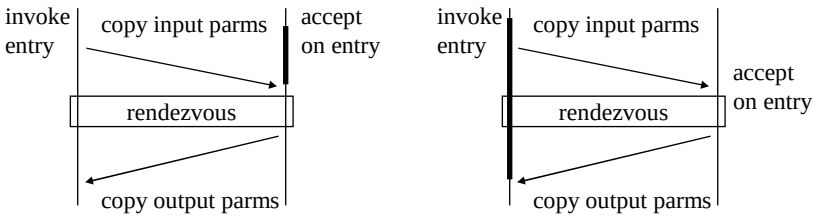
Blocking *vs.* Non-Blocking Primitives

	blocking	non-blocking
send	Returns control to user only after message has been sent, or until acknowledgment has been received.	Returns control as soon as message queued or copied.
receive	Returns only after message has been received.	Signals willingness to receive message. Buffer is ready.
problems	•*Reduces concurrency.*	•*Need buffering:* •*still blocking* •*deadlocks!* •*Tricky to program.*

8

Buffered *vs.* Unbuffered Primitives

- Asynchronous `send` is never delayed
  may get arbitrarily ahead of `receive`.
- However: messages need to be buffered.
- If no buffering available, operations become blocking, and processes are synchronized on operations: **rendezvous**.

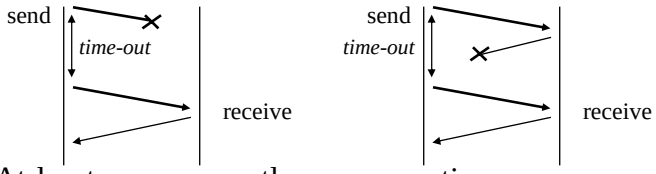


9

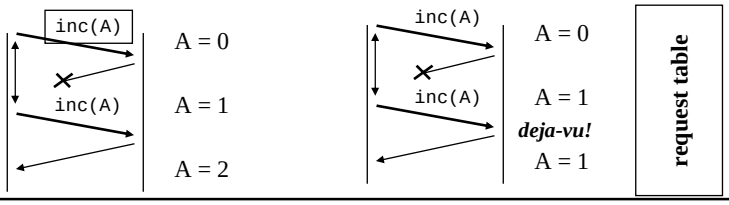
Reliable *vs.* Unreliable Primitives

- Transmission problems:

  corruption loss duplication reordering
- Recovery mechanism: Where?
- Reliable transmission: acknowledgments



- At-least-one *vs.* exactly-one semantics



10

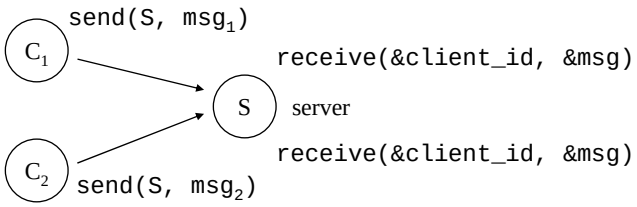
Direct *vs.* Indirect Communication

- Direct communication:

```
send(P, message)
receive(Q, message)
```

- Variation thereof:

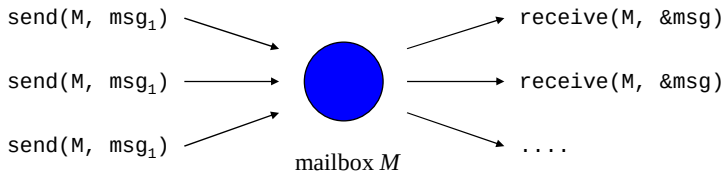
```
send(P, message)
receive(var, message)
```



Direct *vs.* Indirect Communication (cont.)

11

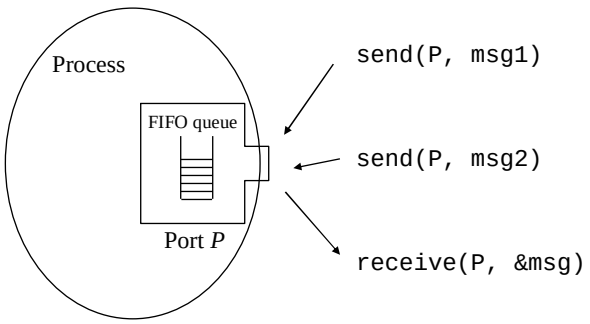
- Indirect communication:
  Treat communication paths as first-class objects.
- Mailboxes:



Direct *vs.* Indirect Communication (cont.)

12

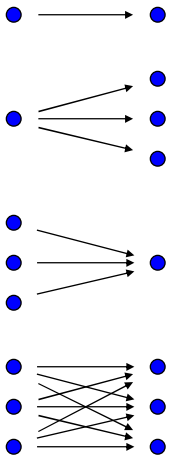
- Indirect communication (cont)
- Ports:
  example: Accent (CMU)



- multiple senders
- only one receiver
- access to port is passed between processes in form of capabilities

13

Communication Schemes



one-to-one unicast

one-to-many multicast

many-to-one

many-to-many

14

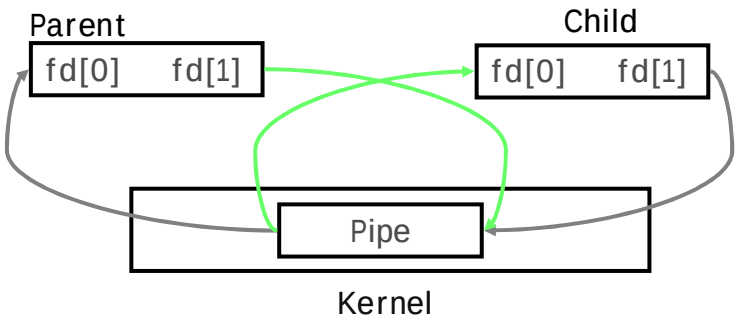
Case Study: IPC on the Same Host

- Ways of Inter-process Communication
  - **Signal**
  - **Passing file descriptor between parent and child processes**
  - **UNIX IPC**
    - ✓ **Pipes**
    - ✓ **FIFOs**
    - ✓ **Stream Pipes**
    - ✓ **Named Stream Pipes**
    - ✓ **Message Queues**
    - ✓ **Semaphores**
    - ✓ **Shared Memory**

15

Case Study: IPC on the Same Host (cont.)

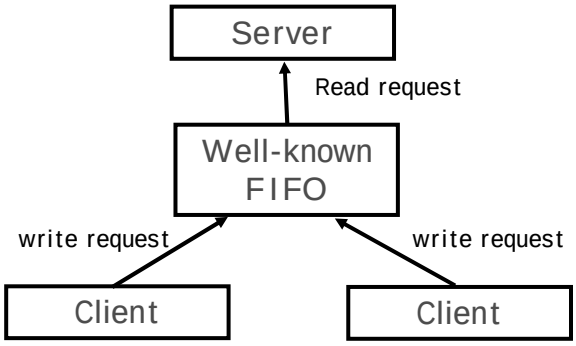
- Pipes
  - **Half-duplex**
  - **Only used between processes that have a common ancestor, e.g., parent and child processes.**



16

Case Study: IPC on the Same Host (cont.)

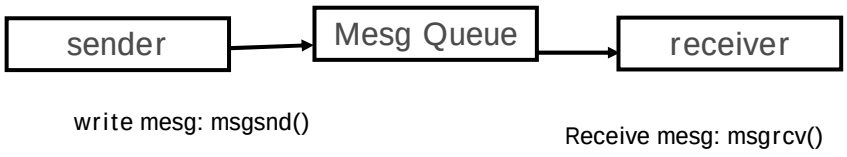
- FIFO (also called named pipe)
  - **Half-duplex**
  - **Can be used between unrelated processes (not necessarily between parent and child processes).**



17

Case Study: IPC on the Same Host (cont.)

- Message Queues
  - **A linked list of messages stored in the kernel and identified by mesg queue id.**
  - **Not necessarily first-in first-out order**
  - **Can fetch messages based on type**
  - **Bi-directional**



18

Case Study: IPC on the Same Host (cont.)

- Semaphore
  - **Not really a form of IPC as pipe, FIFOs, and message queues**
  - **A counter used to provide access to a shared data object for multiple processes**
    1. Test the semaphore that controls the resource
    2. If the value is positive, the process can use the resource and the value of semaphore decrements by one.
    3. If the value is 0, the process goes to sleep until the semaphore value is greater than 0.

19

Case Study: IPC on the Same Host (cont.)

- Shared Memory
  - **Allow two or more processes to share a given region of memory.**
  - **Fastest IPC mechanism**
  - **Synchronization access**

20

Case Study: IPC on the Same Host (cont.)

- Stream pipes
  - **Allow passing open file descriptors between processes (parent and a child)**
  - **Bi-directional**
- Similar to FIFO, we have named Stream Pipe

Remote Procedure Call (RPC)

- We are going to study RPC on Thursday.